Aachen
04. July 2024

Reusable doors in BIM

The integration of reusable doors in digital planning processes

Sophie Elisabeth Drescher
Bachelorthesis 2024

Univ.-Prof. Dr. Jakob Beetz
Design Computation (DC)

Prof. Dr. Linda Hildebrand
Cycle-Oriented Construction

RWTH AACHEN UNIVERSITY

design computation

Abstract

In the building sector, the linear system of processing materials cannot last much longer, as our natural resources are finite and become exhausted. Reusing and utilizing elements for their entire lifespan counters this and lessens the negative effects on the environment. Via building information modeling (BIM), information and potential elements for reuse can be captured in digital models of buildings and used in future projects. However, currently, the majority of the existing building stock is not captured digitally, so to integrate reusable objects in planning and design processes, they need to be digitalized.

This thesis explores methods to do so and aims to automate the creation of a digital model of reusable objects. This is conducted in the form of a case study and is demonstrated using doors that are available in Concular's secondary material shop. Based on the data in Concular, a simplified digital model is created in BlenderBIM. This process is automated via a Python script and results in an add-on for BlenderBIM that is able to automatically create a model of a reusable door. The model is linked with further information in the IFC schema and can be incorporated into BIM software.

The script for the add-on can be accessed on GitHub:

https://github.com/Sophie0o/Doors-for-BIM-

Table of Contents

1 Introduction **4**
1.1 Motivation 4
1.2 Aim 5
1.3 Method 5
2 State of the Art **7**
2.1 Fundamentals 8
3 Analysis **10**
3.1 The data structure in IFC 11
3.2 The data structure in Concular 15
4 Methodology **19**
4.1 Extracting data 20
4.2 Creating the model 24
4.3 Creating a tool 29
5 Conclusion **31**
5.1 Results 31
5.2 Critilcal reflection 31
5.3 Outlook 32
6 References **34**
7 Appendices **38**

1 Introduction

It is evident that our consumptive behavior and especially the exploitative processing of natural resources in the building industry has a devastating impact on our ecosystems. Our resources are finite, and many are already exhausted. The linear processing of them depletes the earth and produces huge amounts of pollution and waste. In fact, "construction and demolition waste (CDW) accounts for more than a third of all waste generated in the EU" (EC, n.d.), and about half of all extracted raw materials are used for construction (EC, n.d.).

Due to its unsustainable character, this take-make-waste principle cannot function in the long run, and it has started to reach its limits (Ruiz Durán et al., 2019). The principle of circular economy counters this by reusing the materials that have already been taken out of the earth. In the building sector, this means that the materials are treated for circulation, and existing buildings are presented as valuable stocks of products and matter (Yang et al., 2022).

1.1 Motivation

The creation of effective material circulation systems via the design of buildings is a significant task for architects and planners. It depends on the reuse of materials at the end of the lifecycle of a building. Coming from many different buildings, reusable materials pose a great variety in their features and condition. There is a lot of information connected to them, such as their properties, their materiality, their producer, date of manufacturing, and their location.

In the design and planning process, information like this is stored using building information modeling (BIM) and open data exchange formats such as IFC, as they allow the digital provision of complex data among many stakeholders. With BIM models being a global standard for the architectural, engineering, and construction industry (AEC), the use of digital models is an indispensable part of the planning process.

So, to integrate reusable objects into the designing process, a digital counterpart of the object is required. However, most of the buildings in Germany were constructed before the official introduction of BIM in 2015 (BMDV, 2015), so most of the existing materials and elements are not digitally documented.

Because of this, the reusable objects need to be recorded and digitalized to be accessible in BIM. The digital counterpart has to be modeled and adapted visually according to the real one in order to represent reliable information (Charef, 2022). However, creating such precise 3D models requires diligent manual labor as well as extensive knowledge. In addition to this, it is usually not just one object but a large number of different components that are to be reused. The manual rendering of this much

data is not only prone to errors but also takes a lot of time, which could be of better use in other parts of the designing process.

1.2 Aim

With our digital tools, we have a great opportunity to improve the sustainable development and implementation of circular economy in the construction industry. By creating smart circular systems, materials, and products can be utilized effectively for long periods of time and thus lower the consumption of resources and the exhaust of emissions caused by their production.

To successfully implement this, the accessibility and exchange of data is indispensable. The key is the link of data from real-life objects to their digital counterparts. In order to be usable in the designing and planning processes, reusable objects need to be accessible in a digital format (Becker et al., 2018).

The goal of this thesis is to develop a strategy to facilitate the integration of reusable objects in digital planning processes. To do so, it explores the steps that are necessary for creating a geometrically and visually accurate digital model of a reusable object and aims to automate this process. This is conducted in the form of a case study based on a product from the secondary material shop Concular.

1.3 Method

The thesis consists of two main parts: an analytical and a practical one. The first one forms the basis by matching data between Concular and the structure of a digital model. The second part explores the realization of the automated creation of the model.

The goal of the first part is to find out which information can be transferred from Concular to the IFC structure and which is needed for the creation of a digital model. To achieve this, the structures and types of information of each of these components need to be analyzed in regard to each other. Overlapping types of information will be explored further to figure out in which form they can be transferred and matched.

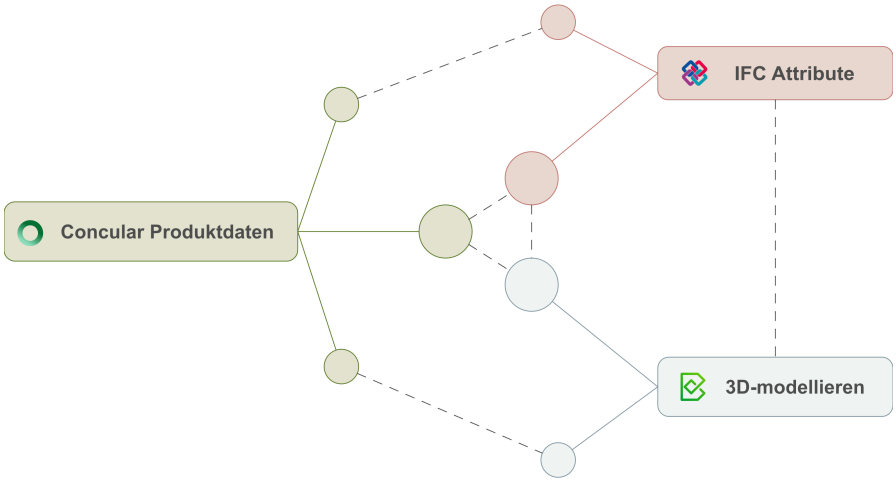


Figure 1: Matching of data between Concular, IFC and BlenderBIM

The results of this form the base of the second part. Here, the focus lies on the processing of the previously defined data and the creation of a 3D object, that is connected to the IFC-structure. The main intent of this part is to automate this process by creating a code that can read out the data from Concular and process it to create a digital model of the selected object in the IFC structure.

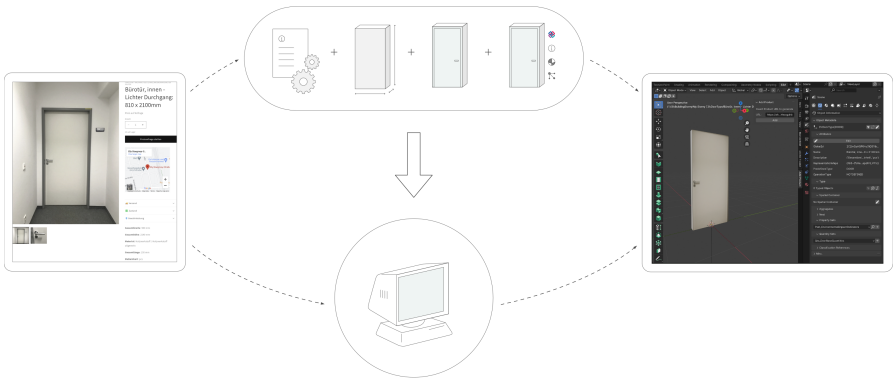


Figure 2: Automating the process from Concular to BlenderBIM

In this part, there are three major functions: the extraction of data from Concular, the creation of the digital object, and the adaption of the process to a tool. They all are executed via a Python script in order to automate the operations. Parts of this code were generated by the AI language model ChatGPT, version 4.0, that was developed by OpenAI. The generated snippets were adapted to fit the specific requirements of the project and integrated into the main script.

2 State of the Art

> *"Effective building component reuse requires specific information about recoverable components. However, 85 percent of the European building stock predates the building information modelling (BIM) technology that stores and links such information."*
>
> *— De Wolf, et al., 2024*

To capture these buildings along with their information, digitization technologies can be used. Currently, the most promising ones are scanning as well as scan-to-BIM technologies such as photogrammetry and LiDAR.

Photogrammetry reconstructs objects based on photos. It recognizes objects based on light rays and can construct two and three-dimensional objects (Pix4D, 2022). Laser scanning technologies, on the other hand, create point clouds that depict the building in a three-dimensional model. At the current state of the technology, the produced data from the reality capturing scans needs to be specified and adapted to match the intended program. Coming from point clouds, sometimes geometries need to be completed and adjusted, and the data needs to be linked to further information for effective integration in BIM. Integrated Data Assessment and Modeling (IDAM) portrays a chance to capture further information like the materiality of the building in the process. However, these methods are yet to be bridged to BIM (Kovacic et al., 2021).

> *"The insufficient automated data transfer from one process step to the next, as well as insufficient data structuring, which would allow a fluently data transfer, is one of the major obstacles to the full exploitation of the BIM potential as a knowledge base for further analysis and optimization, such as material and life cycle assessment."*
>
> *— Kovacic et al., 2021*

All things considered, reality-capturing scans significantly reduce the efforts of manually recovering especially complex products and large amounts of materials. The technology can also be used to improve the connection between digital planning and the is-state of a building and to capture data for possible reusable objects.

That being said, this technology is usually opted for large-scale objects such as buildings and rooms. Capturing smaller individual elements, as they often occur in secondary material shops, with scanning technologies requires a lot of effort and according appliances. In addition to that, the manual processing of the scans is an elaborate but crucial component. This process exceeds the resources of the dealerships, as many already struggle to present each of their products individually online.

2.1 Fundamentals

As this thesis is a case study, it is based on specific tools and data models. The following section briefly explains the fundamentals of

- Concular, which forms the base of the case study
- BlenderBIM, which is used for the creation of the digital model
- IFC, which is used for the interoperability between the model and BIM software

to understand the following processes.

2.1.1 Concular

In Germany, Concular is leading the circular economy in the building industry. It is a digital platform that promotes resource passports for buildings, mediates and sells secondary building materials via a digital database, and also organizes local supply chains. Founded in 2020, the initiative actively works on the sustainable transformation of the building sector (in German: Bauwende) by establishing the circular use of building materials.

Figure 3: Concular logo [concular.de]

> *"In regard of circular economy, material recycling should always be the last option in order to ensure the most sustainable use of resources. … As the emissions of the reuse of building materials is significantly lower, Concular has set itself the goal of reuse in the form of 1:1 substitution and developed the first digital ecosystem for circular construction in Germany. Concular has since become a leader in the field of material passports and the reuse of building materials."*
>
> — *Concular, 2024*

A significant quality of Concular's material shop is the well-curated collection. Each product in the shop is thoroughly examined and presented with additional information. Furthermore, the database of the shop itself is organized in categories such as the product type and the standardized cost group, which facilitates the overview of the products immensely for customers. Because of this, this thesis will conduct the research and digitalization project with Concular as exemplary basis.

2.1.2 BlenderBIM

BlenderBIM is a free add-on for Blender. It combines functions of the 3D creation software with native IFC authoring. Native IFC authoring means having direct access to IFC data and structures, without any software-specific, upstream-placed input templates. This enables the user to not only reference but also edit IFC models, which portrays a huge opportunity for the actual free exchange of data in BIM (Moult, 2021; BlenderBIM).

Figure 4: BlenderBIM logo [blenderbim.org]

Both Blender and BlenderBIM are entirely open source, allowing the user to create individual add-ons and also to program in a scripting environment. Blender's API for Python enables the user to execute commands of the parametric modeling and UV mapping functions via a Python script, directly incorporated in Blender. Because of these functions, this thesis' project of creating a digital counterpart for reusable objects will be based on BlenderBIM.

2.1.3 Industry Foundation Classes

An essential part of the digitalization in the AEC industry is the accessibility and exchangeability of data among multiple stakeholders. However, for optimal results in the different sections, different software is needed. To combine the results and information of each of these parties in BIM, an open data exchange schema is necessary.

A DIN standard for that are the Industry Foundation Classes format (IFC). It is "an open international standard for information used in Building Information Modeling (BIM) that is exchanged and shared among software applications used by the various participants in the construction or facility management industry sector." (ISO 16739-1, 2024). To ensure neutral applicability, IFC defines the structure of the data format for the exchange in BIM.

IFC models can be viewed in different software. "Like PDF, an IFC file is a frozen copy of the original content. It can be viewed, measured, used for clash detection, cost estimation, simulation, and innumerable other uses, but should not be edited." (Baldwin, 2018).

The IFC schema is organized in a layer structure, meaning that the independent core layer defines the superstructure and relations of the IFC schema, while the upper layers can reference them with specific classes.

In this schema

- the identity and semantics (name, machine-readable unique identifier, object type or function)
- the characteristics or attributes (such as material, color, and thermal properties)
- and relationships (including locations, connections, and ownership)
- of objects (like columns or slabs)
- abstract concepts (performance, costing)
- processes (installation, operations)
- people (owners, designers, contractors, suppliers, etc.)

can be codified (buildingSMART, 2022).

3 Analysis

The following part of the thesis will focus on the matching of data of real objects to a digital model and the IFC structure. The analysis of the necessary procedures will be based on a representative case study, focusing on a specific product type from the secondary material dealership Concular.

Concular was chosen because of its outstanding curated and solid structure. The listed products are presented and described in a consequent system; a feature that is crucial for the automated capturing of data. Furthermore, the description of product data is detailed and organized in categories, which facilitates the recognition of the type of information.

The product type was selected in regard to this thesis project, which aims to automate the creation of a digital counterpart of reusable objects. Since this project is based on the following analysis, the case study will concur with the exemplary product type.

For the demonstration, a simple product type that can be handled as an independent object is optimal. Furthermore, the average properties of the type should be common and similar. A product type that fulfills these requirements is doors: Most doors have a rectangular geometry, meaning that they can be simplified to an actual cuboid and still convey important information about their relevant dimensions. In addition to that, doors are an independent system and can be treated as an individual object unit and most of their properties are standardized.

3.0.1 Methodical approach

The first part of this analysis will examine the general structure of an IFC schema to understand how the components function and relate to each other. After that, this chapter presents the specific IFC structure of doors and explains the components that define the attributes of them in IFC.

The next step is the analysis of the product data in Concular. Here, the specific available product information will be examined in regard to the data structure in IFC and the digital model. Overall, this part explores which information can be matched to which attribute of the model and how it can be applied.

3.1 The data structure in IFC

The IFC schema is written in the modeling language EXPRESS and based on STEP – the Standard for the Exchange of Product model data. This provides a standardized structure to be independent from specific software systems.

Roughly explained, the schema defines the structure of building data in a hierarchical layer system. In this system, upper layers can reference lower ones, but not the other way around, ensuring the immutability of the core structure.

In each layer, there are several classes, that are also known as entity types. Each class has attributes assigned to it or an association that connects it to other classes. The attributes and associations can be inherited to further subtypes. For example, an association between an entity of class A and an entity of class B is presented by the class of A having an attribute of the entity of class B (Borrmann et al., 2021). This way attributes of upper classes can be inherited to building elements, that are defined in the IFC schema. With each subclass, the attributes get more specific.

"Some basic characteristics of objects, such as door and wall elements, are stored directly in the schema in the IFC model using attributes in an entity definition. For standard doors, for example, these are the absolute height and width of the door, which can be specified with the OverallWidth and OverallHeight attributes." (Borrmann et al., 2021)

3.1.1 The data structure of IfcDoor

In IFC, a door is a standardized entity like other building elements such as walls, beams, stairs, and windows. It descends from the origin class IfcRoot via the classes IfcObjectDefinition, IfcObject, IfcProduct, IfcElement, and IfcBuiltElement.

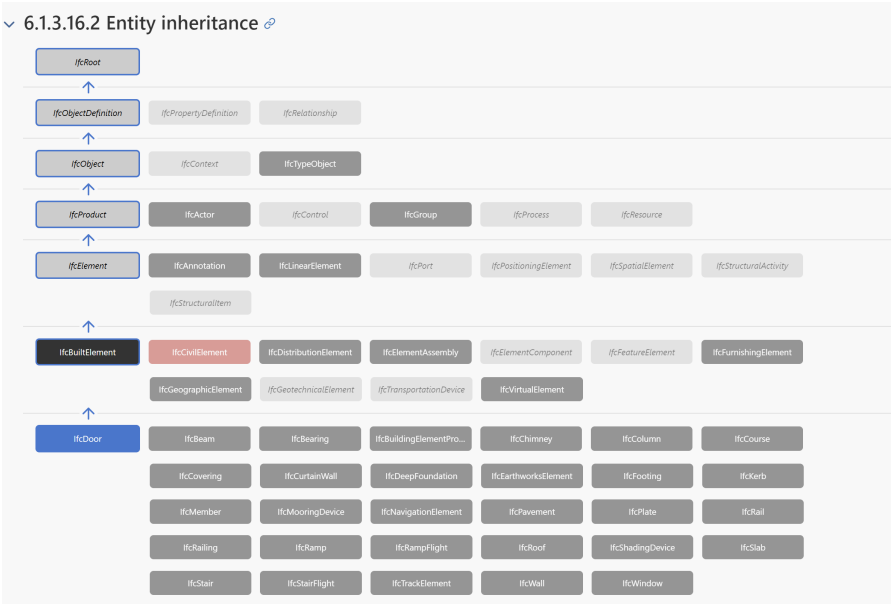
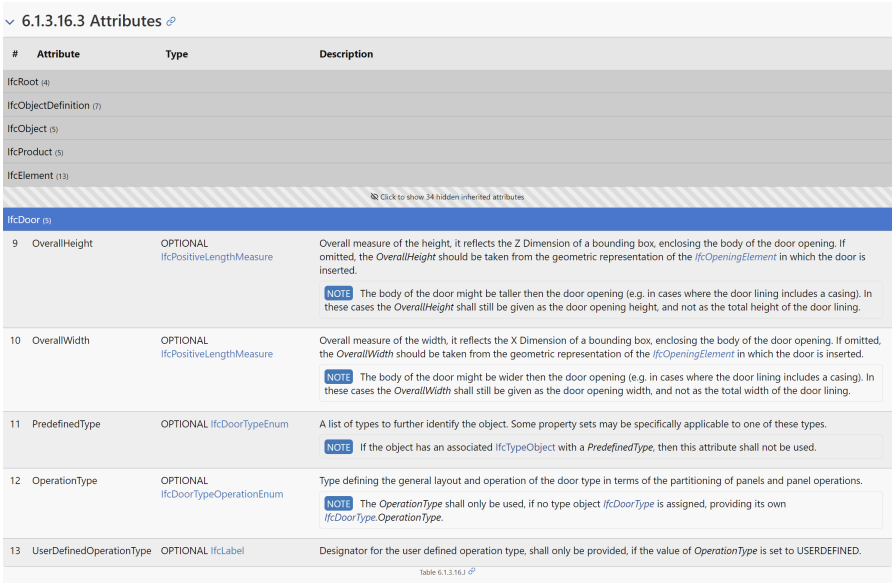


Figure 5: The class structure of IfcDoor [buildingsmart.org]

From this structure, the IfcDoor can inherit a number of attributes, starting with the GlobalID, OwnerHistory, Name, and Description from the supertype IfcRoot. With the class of IfcElement, the door is defined to be a building component and can be assigned to the additional attribute "Tag".

The class of IfcDoor itself contains five attributes that are specific to the definition of a door.



6.1.3.16.3 Attributes

#	Attribute	Type	Description
IfcRoot (4)			
IfcObjectDefinition (7)			
IfcObject (5)			
IfcProduct (5)			
IfcElement (13)			
Click to show 34 hidden inherited attributes			
IfcDoor (5)			
9	OverallHeight	OPTIONAL IfcPositiveLengthMeasure	Overall measure of the height, it reflects the Z Dimension of a bounding box, enclosing the body of the door opening. If omitted, the *OverallHeight* should be taken from the geometric representation of the *IfcOpeningElement* in which the door is inserted. NOTE The body of the door might be taller then the door opening (e.g. in cases where the door lining includes a casing). In these cases the *OverallHeight* shall still be given as the door opening height, and not as the total height of the door lining.
10	OverallWidth	OPTIONAL IfcPositiveLengthMeasure	Overall measure of the width, it reflects the X Dimension of a bounding box, enclosing the body of the door opening. If omitted, the *OverallWidth* should be taken from the geometric representation of the *IfcOpeningElement* in which the door is inserted. NOTE The body of the door might be wider then the door opening (e.g. in cases where the door lining includes a casing). In these cases the *OverallWidth* shall still be given as the door opening width, and not as the total width of the door lining.
11	PredefinedType	OPTIONAL IfcDoorTypeEnum	A list of types to further identify the object. Some property sets may be specifically applicable to one of these types. NOTE If the object has an associated IfcTypeObject with a *PredefinedType*, then this attribute shall not be used.
12	OperationType	OPTIONAL IfcDoorTypeOperationEnum	Type defining the general layout and operation of the door type in terms of the partitioning of panels and panel operations. NOTE The *OperationType* shall only be used, if no type object *IfcDoorType* is assigned, providing its own *IfcDoorType.OperationType*.
13	UserDefinedOperationType	OPTIONAL IfcLabel	Designator for the user defined operation type, shall only be provided, if the value of *OperationType* is set to USERDEFINED.

Table 6.1.3.16.J

Figure 6: The attributes of IfcDoor [buildingsmart.org]

Other attributes can be assigned to the door entity as well. However, due to the limited information that is available in written form in Concular, the project limits itself to the attribute of materiality. "IfcMaterial is the basic entity for material designation and definition; this includes identification by name and classification (via reference to an external classification), as well as association of material properties (isotropic or anisotropic) defined by (subtypes of) IfcMaterialProperties." (buildingSMART, 2024)

- IfcMaterial: basic entity for describing a material.
- IfcMaterialConstituent: Describes the material of a part of a component. The Material attribute itself refers to an IfcMaterial object. The Name attribute is used to uniquely assign the material to the respective part.

There are also other entities for the definition of materials, however, in the case of the exemplary doors, there usually is no more than one layer of materials and the above-named attributes are expected to suffice.

In addition to the static schema of attributes, individual properties can be added to IfcDoor. In BuildingSmart, there are various templates that cover among others the properties of condition and environmental impact indicators, as well as common information specific to doors.

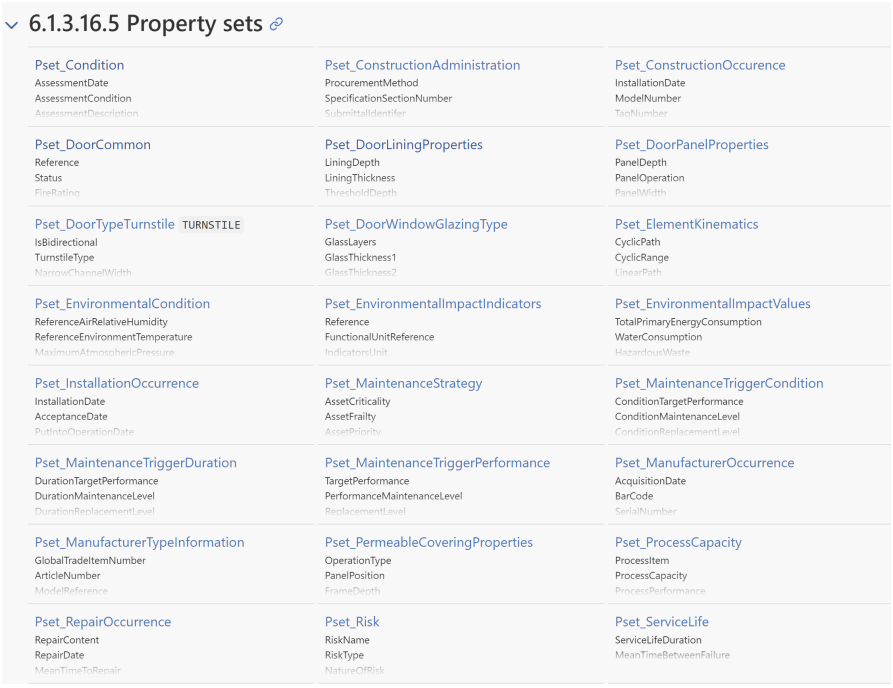


Figure 7: PropertySet templates for IfcDoor [buildingsmart.org]

3.2 The data structure in Concular

In Concular, the product is described by a number of individual categories, containing general information about properties and dimensions. To match this information to the IFC attributes, they have to be filtered, sorted, and allocated.

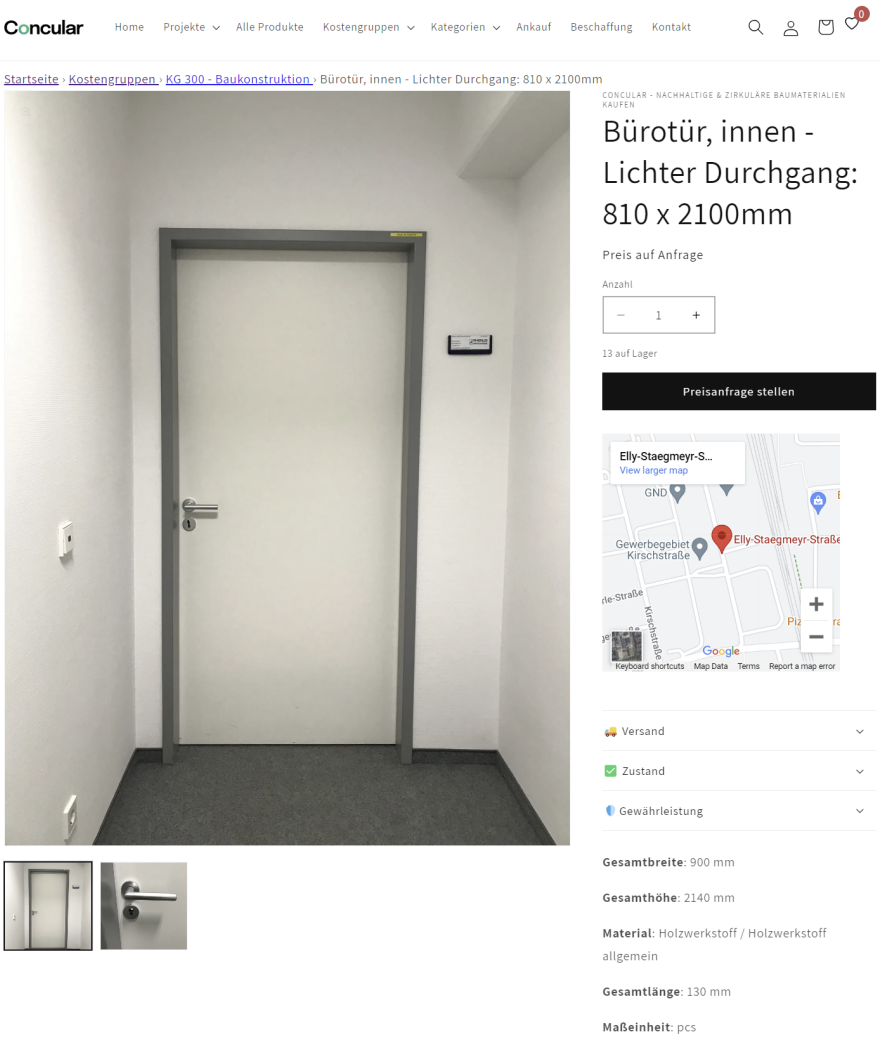


Figure 8: Door listing in Concular [concular.de]

Two pieces of information that can be found in every single product are its name and description. Both of them correspond to IFC attributes of the origin class IfcRoot, with the name being an IfcLabel and the description being an IfcText. The two attributes can be inherited to subclasses of IfcRoot, thus also connecting to the entity IfcDoor.

In general, with every subclass, new attributes can be allocated and inherited. The entity IfcDoor therefore has several specific attributes connected to it. Similarly, in Concular the description of the product also contains a number of specific information.

Analyzing the data in the description of a product shows that each product has an individual set of categories to describe it. Some products contain detailed information about geometries and dimensions as well as about the materiality and producer, whereas others have only information about the geometry. The following table presents examples of the differences in the product descriptions.

Office door anthracite 1050x2150x150	**Office door, interior - light passage: 810 x 2100mm**	**Wirus T30 fire door 1000x2140x170**
Overall width: 1050 mm	**Overall width:** 900 mm	**Overall width:** 1000 mm
Overall height: 2150 mm	**Total height:** 2140 mm	**Overall height:** 2140 mm
Overall length: 150 mm	**Material:** Wood-based material	**Overall length:** 170 mm
Clearance dimensions: 920 x 2090	**Overall length:** 130 mm	**Clearance height:** 940 mm
Clearance dimensions: 910 x 2080	**Unit of measurement:** pcs	**Clear width:** 210 mm
Colour: Anthracite		**Material:** Wood-based material
DIN direction: DIN R	Clear passage: 810 x 2100mm	**DIN direction:** DIN R
Material: Wood-based material		**Year of construction:** 1998
Quantity: 2 units		**Other:** various fittings
		Quantity: 2 units
Removal will take place in June 2025.		The extension will take place in June 2025.

Figure 9: Descriptions of different doors from Concular (translated by the author)

Nonetheless, there are common denominators in the descriptions of the products. The most apparent one is that in almost every product description, some variation of the dimension-category can be found. This can be explained by the fact, that doors can be easily measured by anyone without the need of too specific tools.

Another category that appears quite often is the materiality. Less often found categories would be the year of manufacturing and the producer. Because of the availability of certain categories, this project will focus on incorporating the dimensions and materiality in the automated model for demonstrative purposes. Other categories could be added later on in the attributes, but to demonstrate the integration of real object data into BIM, the dimensions and the attribute of materiality are used as representatives.

The amount and detail of information in the description of secondary materials varies from product to product. For the actual matching of this data to an IFC structure, the amount of information is not the deciding part but rather the structure of the data. The principle is the same as it is in the IFC schema. The standardized structure and definition of terms and attributes ensure neutrality and universal accessibility. So, in order to match data of reusable products to the IFC structure, a consequent organization of labels and categories is essential.

3.2.1 Matching dimensions

In the product description, the categories that show the data of dimensions always include a height, length, and thickness. This is crucial for the creation of a three-dimensional model, as the height portrays its scale along the z-axis, the length the scale along the y-axis, and the thickness the scale along the x-axis.

These three measurements, however, are not all that there is to the dimensions of a door. Some of the doors are sold with a frame or a top window, which are also included in the description of the product. Furthermore, the label of each measurement is not uniform, meaning that there are different words used for the description of a unit. Sometimes the thickness is labeled as depth, then there is a category labeled "height", a category labeled "overall height" and a category called "door height" in one product description.

Because of this, there is often no single category for height, length and thickness, but several categories describing each of the dimensions.

Office door anthracite 1050x2150x150	**Office door, interior - light passage: 810 x 2100mm**	**Wirus T30 fire door 1000x2140x170**
Overall width: 1050 mm	**Overall width:** 900 mm	**Overall width:** 1000 mm
Overall height: 2150 mm	**Total height:** 2140 mm	**Overall height:** 2140 mm
Overall length: 150 mm	**Material:** Wood-based material	**Overall length:** 170 mm
Clearance dimensions: 920 x 2090	**Overall length:** 130 mm	**Clearance height:** 940 mm
Clearance dimensions: 910 x 2080	**Unit of measurement:** pcs	**Clear width:** 210 mm
Colour: Anthracite		**Material:** Wood-based material
DIN direction: DIN R	Clear passage: 810 x 2100mm	**DIN direction:** DIN R
Material: Wood-based material		**Year of construction:** 1998
Quantity: 2 units		**Other:** various fittings
		Quantity: 2 units
Removal will take place in June 2025.		The extension will take place in June 2025.

Figure 10: Different descriptions of doors from Concular (translated by the author)

To ensure the proper representation of the size of the object, the outermost dimensions are decisive. As a result, the biggest values of each category that describe height, width or thickness are to be assigned to form the geometry of the digital model.

3.2.2 Matching materials

For the material, this system cannot apply. The dimensions are described with numerical values, whereas the material is described in a textual value, so there is no "biggest value". Instead, all given materials are listed and assigned to the attribute IfcMaterial.

4 Methodology

The following chapters focus on the practical implementation of the processing of data and the creation of the model. The execution of this consists of two main functions: one extracts, filters and defines the information from the Concular shop, whereas the other one creates a model based on this data. The two functions conclude in a BlenderBIM add-on, that allows the creation of the digital model just by entering the URL of the product.

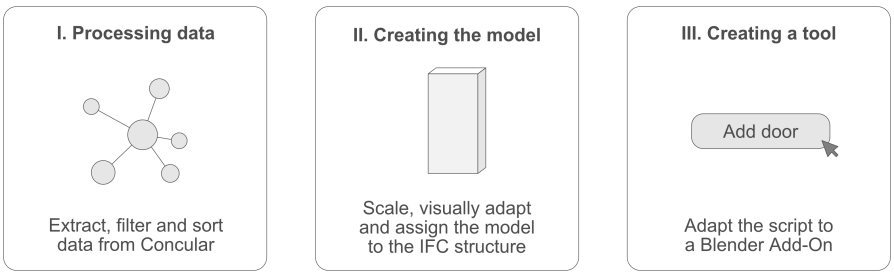


Figure 11: Steps of the process

All of the functions are executed via a Python script. The first function is based on a scraping operation, that reads out the data of the product website. Algorithms filter this data and define the results as variables in the code, which allows to reference them in further operations. So, the variables represent the information that is contained in the digital model of the door.

The model itself is created in BlenderBIM, as this program combines the 3D modeling features of Blender with native IFC-authoring. This allows to model an object and integrate it directly into the IFC schema, together with the information that is linked to it. The modeling operations of this step are executed via a Python script in Blender's scripting environment. Because of this, the scripts of the two functions are compatible and can be combined into a single script that coherently executes all defined operations.

4.0.1 Methodical approach

Because of the consistent structure of Concular's product websites, the operations for the extraction of data run in the same way for every door that is listed in Concular. In fact, the only thing that is subject to change in the input of the code is the URL of the product. The result from this function is a coherent list of defined variables that contain specific information from Concular.

Initially, the entirety of this first function was executed via a Python script in VisualStudioCode, with the results being listed in a locally saved CSV file. The CSV file was then referenced by the code of the second main function in Blender. However, in order to let the code run independently

from any local savings, the interim step of the CSV file was eliminated by importing the first functions to the Blender script. Thereby, the results can be referenced directly in a coherent code that includes this main function together with the following code that creates the 3D model.

4.1 Extracting data

To read out the data from Concular's product pages, a code needs to access the website and extract specific elements of it. By inspecting the website, the Xpath of each of these elements can be found. The Xpath describes the location of an element in the structure of the website, which means that it directly points to the element and its content. In a scraping code, the Xpath along with the URL of the website, can therefore read out specific elements of a website (W3Schools, 2024).

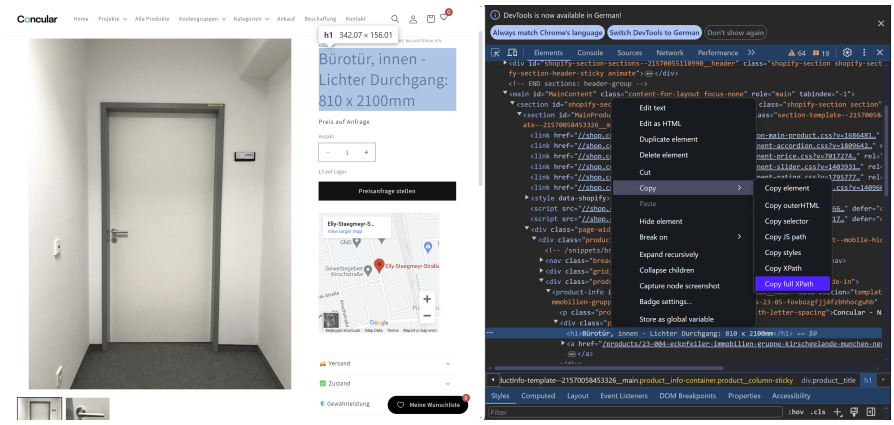


Figure 12: Xpath of the name [concular.de]

On Concular's door product page, there is an element for the name, an element for the description of the product and an element for images. Since Concular's website structure is consistent, the specific Xpaths of each element apply for all product websites. Because of this, each type of information can be extracted with a scraping code, that reads out the according the element.

For example, the name element of the product has its own specific Xpath, so the code can directly access this information and extract it without any further ado. The extracted information then gets defined as "product-name" and can be referenced as such.

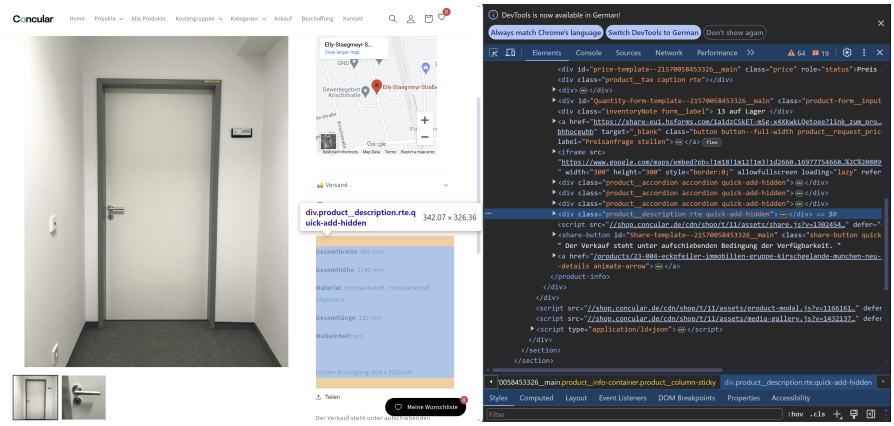


Figure 13: Element of the product description [concular.de]

Similarly, the data for the description can be extracted via the Xpath from the description element in Concular. This way, everything that is listed in the description of the product in Concular can be referenced as the description of the product. In this case, however, there are sometimes slightly different variations of the Xpath for the description element from website to website. Because of this, the code includes the different variations of the Xpath to ensure the data of the description is found. The extracted information gets defined as "product-description" in the form of a list of values.

Based on this list, further specific data can be read out, as the description of the product contains all previously defined relevant information. The structure of the description in Concular lists a number of categories such as "Gesamthöhe" (overall height), "Gesamtbreite" (overall width), "Gesamtlänge" (overall length) and "Material" (material). These categories are presented as terms, and to each of them, a numerical value or text is allocated. This means that the terms define the type of information, so the allocated values and texts are the actual content.

To figure out the data for the dimensions of the object, an algorithm searches the product-description list for all terms that include the word for each dimension. As a result, it creates a list of all categories that include the term "höhe" (height), a list of all categories that include "breite" (width) and a list that includes all categories that include the term "länge" (length). However, it has become apparent that some products use the term "tiefe" (depth) instead of "länge" (length) for the thickness of an object. To make sure that the values of these categories are always included, a fallback loop repeats the operation if there is no term "länge" found and instead creates a list of the categories that include the term "tiefe".

As a result, there is a list of categories and values for each axis of the dimension of the object. To find the maximal values for each axis, another algorithm searches each list and filters out the maximal numerical value. The results of this are then defined as "max-depth", "max-width" and "max-height".

For the data of the material a similar approach is used, meaning that the algorithm searches the product-description for the term "material". Here, however, the results are not filtered but instead listed together under the definition of "product-material".

4.1.1 Processing images

The last information that needs to be extracted from the website is not textual but a visual one. In order to give an accurate representation of the door, its looks also need to be anchored in the digital model.

To depict the appearance of the door, an image of the product can be projected onto the digital model. For this to be possible, an image of the door needs to be downloaded and edited to depict only the door.

With the Xpath of the image section of Concular's website, the main image, the one that usually shows the door as a whole, can be downloaded and saved to the memory. This image then needs to be processed with OpenCV, which can be installed with the command "pip install opencv-python" in the terminal. OpenCV is an image recognition and processing program that can be incorporated directly into a Python code (OpenCV, 2024). With this installed, the program can read an image and, in this case, search for the largest trapezoid.

To improve the image processing, the original images need to be sized down, blurred and their contrast needs to be increased for the program to find lines between contrasting surfaces.

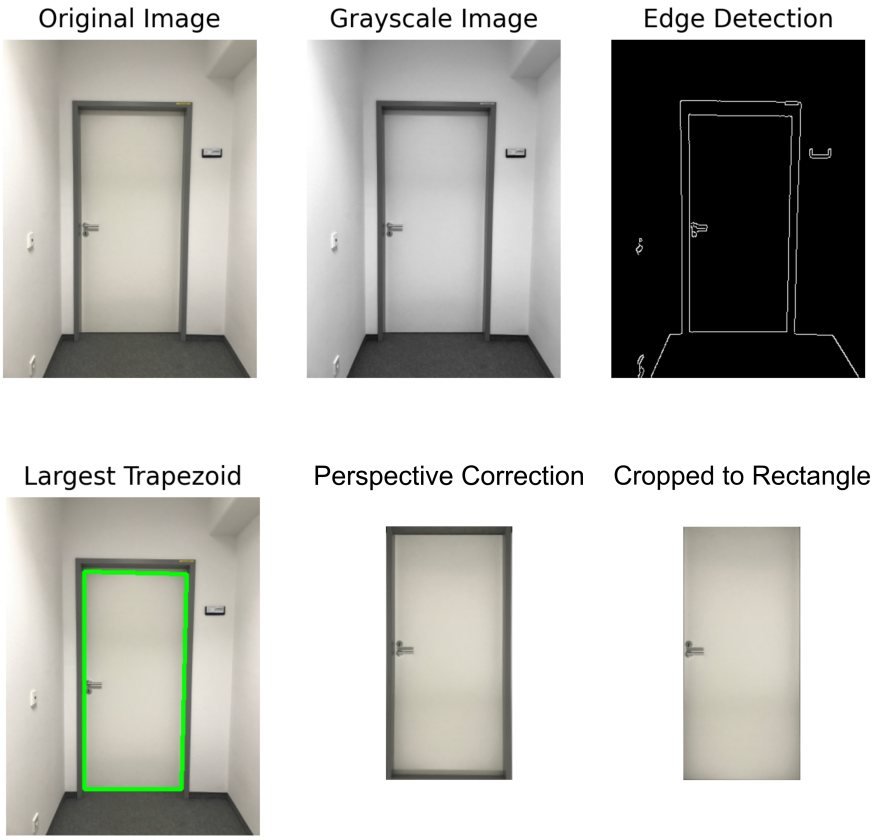


Figure 14: Steps of the image processing

This method works well for images with clear lines and little distortion, but still not for all doors in Concular. To further increase the visual capturing of the them, one approach was to search for the two longest vertical lines and span a plane in between them to find the trapezoid surface of the door. Another one was to search for three lines that intersect in two points, which should represent the frame of the door. However, neither of these approaches increased the visual recognition of the doors, so the script continues with the operation that searches for the largest trapezoid.

Once the trapezoid is found, the program corrects the perspective of the found trapezoid to form a rectangle and crops the image along the outlines. As a result, the cropped image can be accessed in the code via "output-cropped-path", just like any other information that has been defined in the code.

4.2 Creating the model

The following module is about the creation of a digital model based on the previously extracted information. Since the model is created in Blender, the following operations are Blender commands, that are executed via the Python script.

For the creation of an IFC model, it is crucial to work in an IFC project in BlenderBIM. To ensure working in an IFC environment, the code therefore makes sure that the BlenderBIM add-on is enabled and a new IFC project is created to work in once the code runs for the first time.

In BlenderBIM's IFC projects, there are various architectural elements predefined and there is also a default door model. This model is detailed with a frame, doorknob and even 2D plans connected to it, so it makes a great base for architectural use.

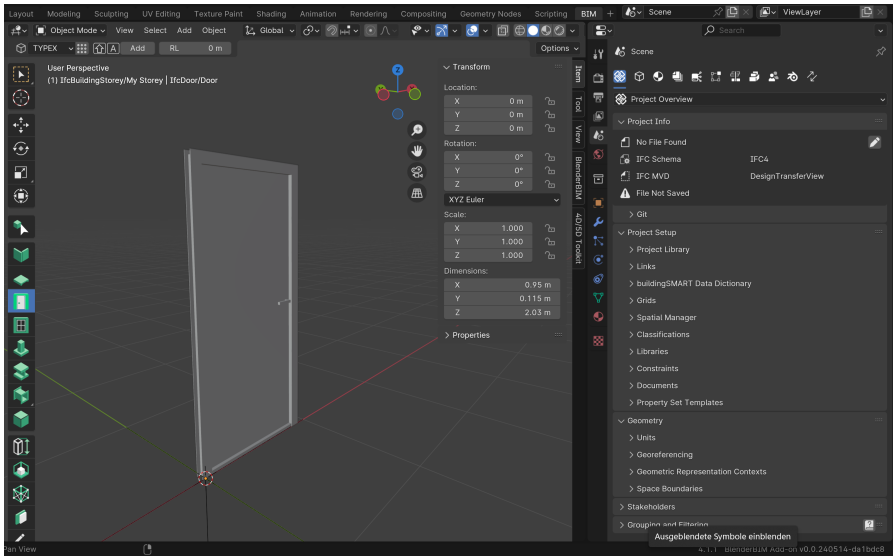


Figure 15: BlenderBIM: Default IfcDoor in BlenderBIM

However, regarding the available data from Concular, these features portray a disadvantage. The product description in Concular does not provide enough information to accurately determine all properties of the default model. Because of this, there is a high chance of having incorrect information from the default model assigned to the new door model. In addition to that, the complex geometry of the default door also distorts the projection of the image onto the door. Because of the various possibilities for inaccuracy and false information, a simplified model that represents the overall dimensions in the form of a cuboid is chosen as the base instead.

In the script, the operations to create the model are linked with the previously defined variables. This way, the door gets adjusted to the content of the variables, but the script itself and the operations do not change. Due to this structure, there is no need to manually adjust the script for

different doors.

This means, that the properties of the model essentially depend on the first main function, which filters and defines the extracted information. To add more information to the model, a new variable needs to be created and referenced in the according Blender operation. This principle also applies to the information represented in the IFC structure of the model. The variables that define information are linked with the according attribute and are adjusted automatically.

4.2.1 The base model

In the IFC environment, the 3d modeling still follows original Blender operations. The base of the model is formed by the standard cube, so the first command is to add it to the scene. The cube then gets placed to the origin of the coordinates, so that the ground surface aligns with the x- and y-axis.

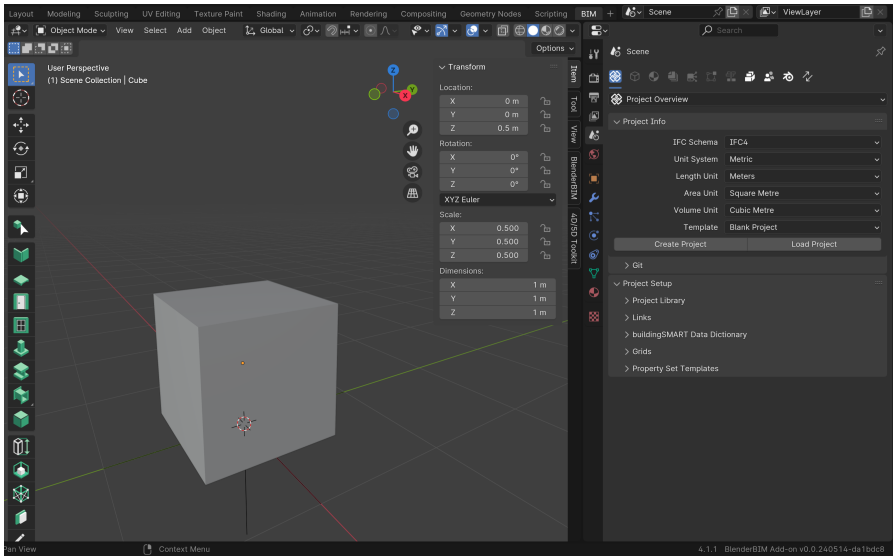


Figure 16: BlenderBIM: Cube on origin point

After this, the cube gets scaled based on the filtered information about the dimensions of the door. For that, the resulting maximum value of each dimension category is allocated to one axis. This means that max-depth presents the scaling along the x-axis, max-width presents the scaling along the y-axis and max-height the z-axis. So now, the model already represents the maximum dimensions and the space of the actual door.

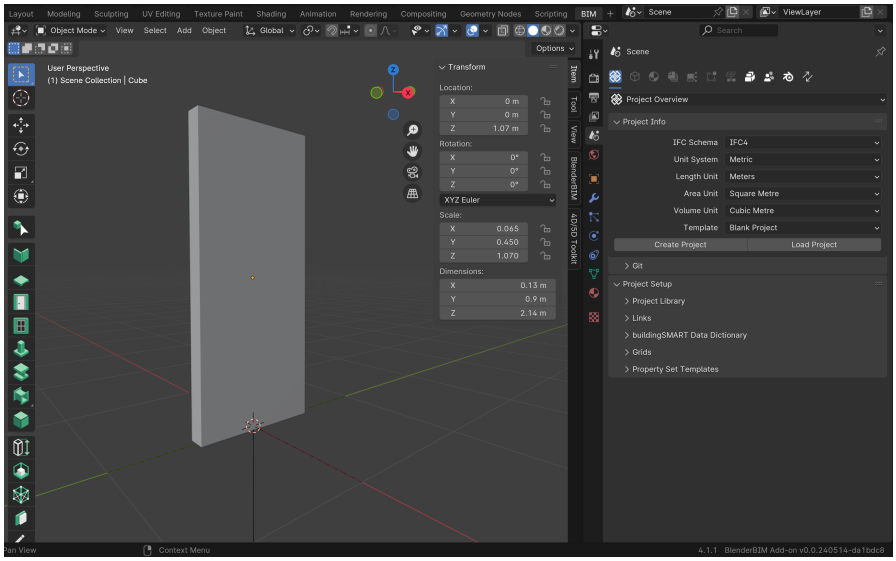


Figure 17: BlenderBIM: Scaled base model

In addition to that, the label of the model gets set to equal the variable product-name, so now the cube's name is what was previously defined the be the product-name in the scraping code.

4.2.2 Visual representation

With this basis and the shape of the model being set, the image of the door can be projected onto it. The blender operation for that is called UV mapping, which means that the surfaces of an object get unwrapped flat and form so-called UV maps. The UV maps can be laid onto a texture and project the image beneath them onto the model. To achieve this for the door model, first, the cropped image needs to be set as a new texture material for the active object.

At this stage, normally, all faces of the object would be placed on this image if it gets unwrapped. However, for the door model, only the largest two faces need to show the image in its entirety. To achieve this outcome, the largest face, so the one whose normal vector points along the x-axis, needs to be selected first and then projected onto the texture with the command "scale to bounds". This operation automatically aligns the outline of the selected UV map with the outline of the image and therefore projects the whole image on the selected face.

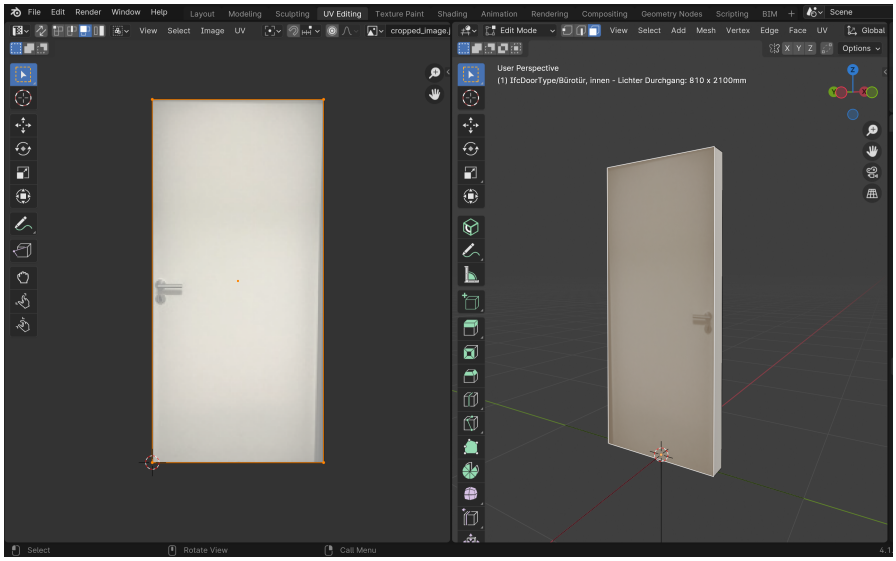


Figure 18: BlenderBIM: UV mapping of the image onto the model

The same needs to happen for the other side of the object too. However, since the two faces mirror each other, the operation cannot be applied for both at the same time but instead has to be executed individually. This allows to mirror and adjust the UV map of each side accordingly and ensures the correct projection of the image onto the object.

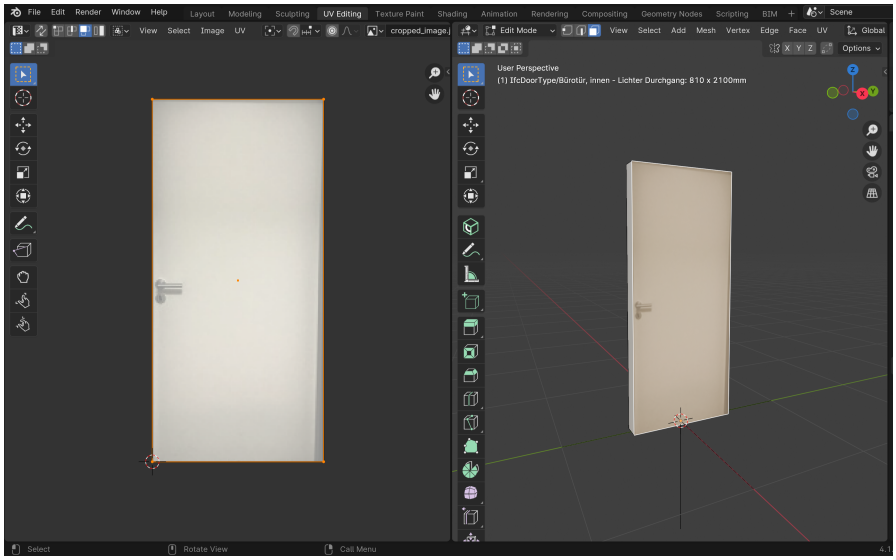


Figure 19: BlenderBIM: UV mapping of the other side of the model

4.2.3 Implementation in IFC

With these steps being complete, the digital model already represents the door in its geometry, looks, and name, but it does not have any relation to the IFC structure yet. To integrate the created object into the IFC schema, it needs to be assigned to an IFC class – in this case the IfcDoorType.

With a simple command, the object can be assigned to this class, together with the predefined type "door". This command automatically connects the object to the IFC-structure of IfcDoorType and allows to connect the according attributes to it as well.

The remaining information can now be set as attributes and are connected to the digital model. For example, the name of the door is now automatically transferred to the IFC-name-attribute. The earlier defined list product-description can be matched to the description attribute of the object and another command allows to set the product-material value to the material attribute.

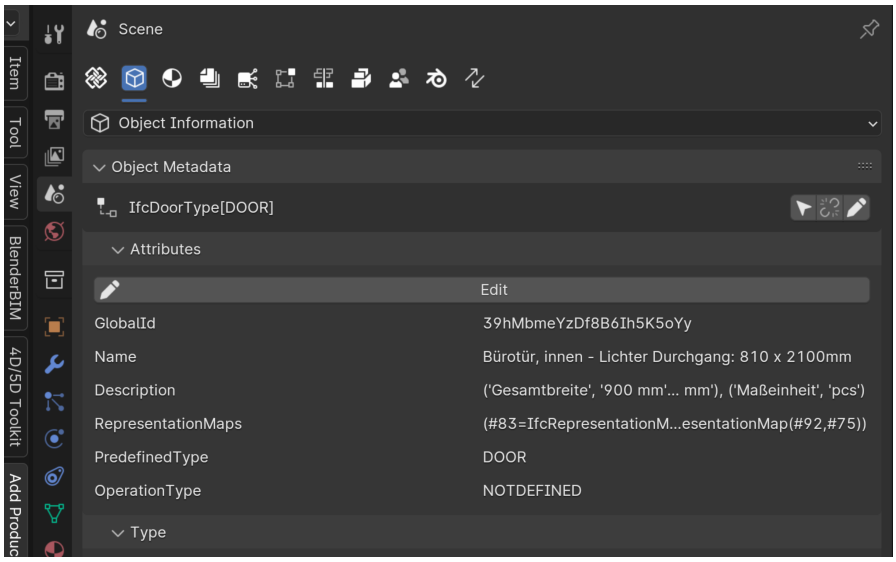


Figure 20: BlenderBIM: IfcAttribute: Name; IfcAttribute: Description

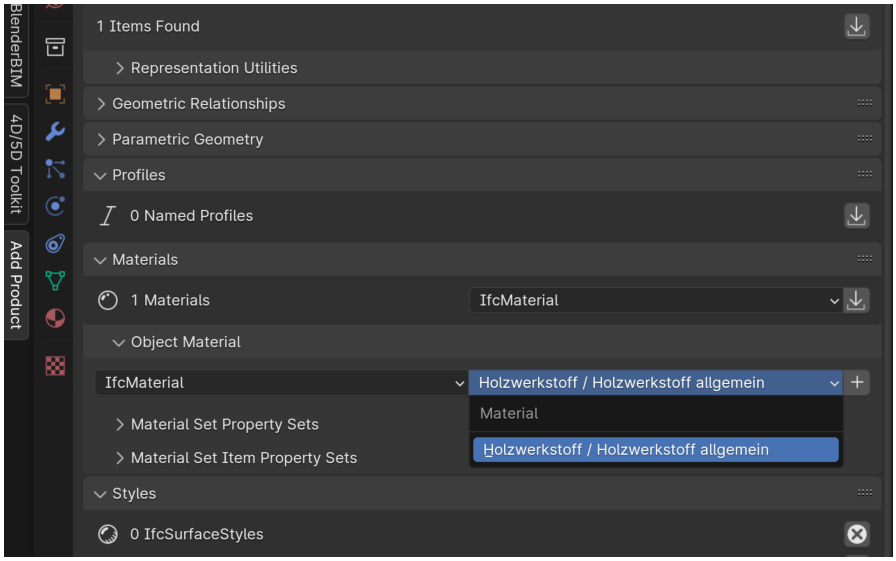


Figure 21: BlenderBIM: IfcAttribute: Material

The operation to add other attributes is the same as it is for the description and the material, so further attributes can be added as well. They

only have to be defined first, during the scraping of data, and then be accessible as a variable.

4.3 Creating a tool

Since the overall goal of this thesis is to facilitate the integration of reusable objects in BIM, the major part focuses on the automation of the creation of digital models. However, in the end, the actual usage of it also depends on the usability for users.

Generally, the combined code can be executed for any door from Concular. To change the door to be created, the URL of the product has to be replaced in the code in Blender's scripting environment. To make this process more accessible for users, an add-on with a user interface in the standard modeling layout can take this on. The adaption of the main script to an add-on in the standard 3D viewer of BlenderBIM allows the user to execute the script without entering the code. This way, the automated creation of the doors becomes simple and intuitive to use for people who are not familiar with coding.

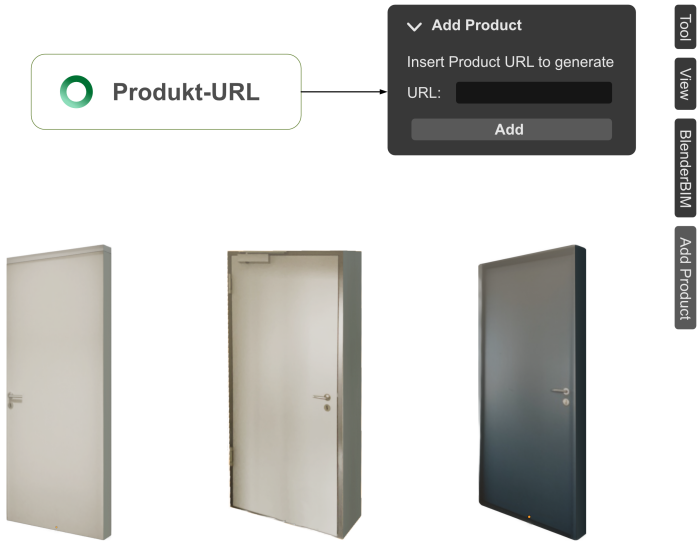


Figure 22: Function of the Add-On

4.3.1 Creating an add-on

The structure of the code is static, so the operations for the creation of each door are essentially the same, with the exception of the content of the information. Because of this, the only thing that represents each door individually in the input of the code is the URL of the product's website. This means that, in order to change the door to be created, only the URL of the product needs to be changed in the code.

This fact can be incorporated into the add-on so that only the URL of the product has to be pasted into the user interface to create the door auto-

matically. To create this add-on, another Python script can be used. Via the script, the position and the content of the add-on can be set, which also means that the existing script can be linked to it. (CG Masters, 2020) So, the add-on can be defined to appear in Blender's 3D viewer as a new panel that says "Add Product." In addition, the content of the add-on can be determined to display the instruction "Insert product URL to generate", an input field, and a button to run the code.

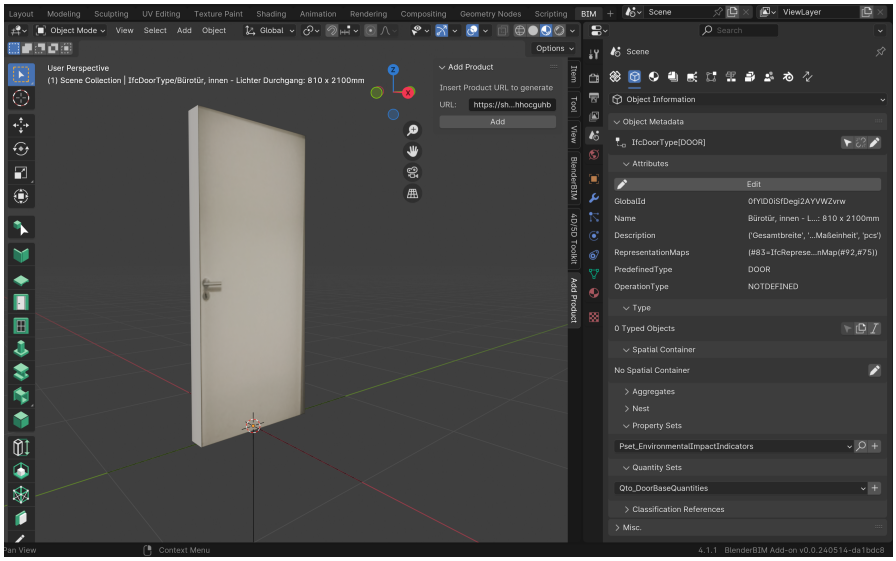


Figure 23: BlenderBIM: Add-On and automatically created model

To connect this add-on with the existing script of the main function, the text that can be inserted into the input field needs to be defined as a variable. This way, it can be permanently referenced in the code while containing the flexible product URL. As a result, the code of the main function can be embedded in the add-on and connected to the inserted URL.

5 Conclusion

The thesis demonstrates a prototypical approach to automate the creation of a digital model of a reusable door. The project is conducted as a case study based on the doors available in the secondary material dealership Concular. Because of this, the methods are tailored to fit the data structure of Concular and cannot be applied universally. Nonetheless, the overall structure of this project can be adjusted to fit similar types of websites, provided that there is also a coherent structure in their data.

5.1 Results

The major result of this thesis is that with a coherent structure, textual information can be easily connected to an IFC attribute in BlenderBIM. The actual difficulty in the creation of the digital model, however, lies in the precise geometric representation. There is simply not enough information about this in the textual description of the doors. Instead, a great part of this information is conveyed via the images. They present the actual looks, the layout, the opening directions, and much more information visually, but this data simply cannot be read out with a code. Therefore, this cannot be modeled automatically.

Because of this, the model has to be simplified and present only the overall shape and space of the product to avoid conveying incorrect information. The projecting of the image of the door attempts to connect the visual information with the model, but at the bottom of the line, this is not enough for architectural plans.

So, overall the thesis shows that a simplified version of the object can be created automatically. It is not a detailed and very precise depiction, but it can be linked with additional information and represent various properties of the door. As a result, it can be integrated into the IFC structure and therefore also be used in BIM.

5.2 Critilcal reflection

The thesis explores methods for the automated creation of digital counterparts of reusable objects and develops a prototypical tool for that. The result is a demonstrative and exemplary strategy that is executed via a Python script in BlenderBIM. It successfully reads out data from Concular and translates it into a 3D model that represents a reusable door visually and in its overall geometry. In addition, the model can be used in BIM software, which is the overall goal of this thesis. As it is an exemplary tool for demonstrative purposes, it focuses on realizing the main function. This way, other features can be added and incorporated following the same strategy and structure.

For one thing, more information can be added to the model. For example, the description of the doors in Concular often also includes the pro-

ducer, the year of production, and many other details. Information like that could be defined in the script and added to the IFC attributes of the model to be included in BIM.

Another component that needs to be improved is the image processing. It does work for simple images, in which the doors have a great contrast to their surroundings and clear outlines, but it fails to recognize doors in less ideal images. Especially images of doors with glass elements and unclear outlines cannot be captured and are therefore projected incorrectly onto the model. An approach to counter this could be the incorporation of artificial intelligence. With image recognition software that has been trained on doors, the capturing of the doors could be more precise and even work on more complex door types. It could supersede the current image processing and still fit well into the overall structure of the script.

Based on this, further features could be added as well. With the extraction of visual information, different types of doors could be recognized and translated into more specific default models. This way, the models could be connected to 2D plans and represent the model more accurately in BIM software.

5.3 Outlook

The main aspect of the developed method is that the automated creation of the model allows to quickly provide a digital representation of the door. This can be used to represent reusable doors in digital designs and to easily test out different ones. In addition to that, the doors can be included in BIM models, which also means that they can be digitally accessed again and again.

This method of automatically creating digital models of reusable objects portrays a potential to improve the capturing of reusable stock. Combined with a standardized template of necessary and additional input information, this method can quickly produce a digital model of reusable objects. This is especially relevant for smaller and simple objects that often appear in larger quantities, such as windows and doors. Elements like this are hardly ever captured by scanning, let alone transformed into digital models, even though they make up a considerable part of the stock of reusable materials.

With the developed method of this thesis, these elements can be captured without any need for special devices. This means that the reusable objects can be digitalized easily, and even though they are only simplified versions, they can be integrated into the planning process.

With the incorporation of artificial intelligence in the image recognition module, there is a chance to read out further information from the images. For example, it could capture whether the doors are single or double-wing and in which direction they open. With information like that, the digital model could be more precise. Different default models

could be used to present the individual types of doors instead of the simplified cuboid for all. This way, they could be connected to schematic 2D drawings and, therefore, be used in architectural plans as well.

So, with improved features, the developed strategy portrays a chance to digitally record reusable objects and integrate them in the digital planning process.

6 References

Baldwin, M. (2018). *What is IFC?*. BIM Connect.
https://bimconnect.org/en/software/what-is-ifc/ (Accessed on: 2024, 1. July)

Becker, R., Falk, V., Hoenen, S., Loges, S., Stumm, S., Blankenbach, J., Brell-Cokcan, S., Hildebrandt, L., & Vallée, D. (2018). BIM – Towards the entire lifecycle. *International Journal of Sustainable Development and Planning*, *13*(01), 84–95.
https://doi.org/10.2495/SDP-V13-N1-84-95

BlenderBIM. (2022, 7. May). *Introduction to BIM*
https://docs.blenderbim.org/users/introduction_to_bim.html (Accessed on: 2024, 1. July)

Borrmann, A., König, M., Koch, C., & Beetz, J. (Eds.). (2021). *Building Information Modeling: Technologische Grundlagen und industrielle Praxis (2. Auflage).* Springer Fachmedien Wiesbaden.
https://doi.org/10.1007/978-3-658-33361-4

buildingSMART. (2024, 23. May). *IfcDoor*. GitHub.
https://github.com/buildingSMART/IFC4.3.x-development/blob/master/docs/schemas/shared/IfcSharedBldgElements/Entities/IfcDoor.md (Accessed on: 2024, 1. July)

buildingSMART. (2024, 23. May). *IfcMaterial*. GitHub.
https://github.com/buildingSMART/IFC4.3.x-development/blob/master/docs/schemas/resource/IfcMaterialResource/Entities/IfcMaterial.md (Accessed on: 2024, 1. July)

buildingSMART. (2022, 29. July). *Industry Foundation Classes (IFC)*. GitHub.
https://github.com/buildingSMART/technical.buildingsmart.org/blob/main/Industry-Foundation-Classes-(IFC).md (Accessed on: 2024, 1. July)

Bundesministerium für Digitales und Verkehr. (2015). *Stufenplan Digitales Bauen: Einführung moderner, IT-gestützter Prozesse und Technologien bei Planung, Bau und Betrieb von Bauwerken*.
https://bmdv.bund.de/SharedDocs/DE/Publikationen/DG/stufenplan-digitales-bauen.pdf?__blob=publicationFile (Accessed on: 2024, 1. July)

CG Masters. (2020, 10. July). *Data Visualization in Blender and Python.* YouTube.
https://www.youtube.com/watch?v=Xrixs_XuDQo&t=1382s (Accessed on: 2024, 1. July)

Charef, R. (2022). The use of Building Information Modelling in the circular economy context: Several models and a new dimension of BIM (8D). *Cleaner Engineering and Technology*, *7*, 100414.
https://doi.org/10.1016/j.clet.2022.100414

Concular. (2024). *Circular Economy in der Baubranche*. Concular.
https://concular.de/circular-economy-in-der-baubranche/ (Accessed on: 2024, 1. July)

De Wolf, C., Çetin, S., & Bocken, N. M. P. (Eds.). (2024). A Circular Built Environment in the Digital Age. Springer International Publishing. https://doi.org/10.1007/978-3-031-39675-5

European Commission. (n.d.). *Buildings and construction*. https://single-market-economy.ec.europa.eu/industry/sustainability/buildings-and-construction_en (Accessed on: 2024, 1. July)

European Commission. (n.d.). *Construction and demolition waste*. https://environment.ec.europa.eu/topics/waste-and-recycling/construction-and-demolition-waste_en (Accessed on: 2024, 1. July)

ISO 16739-1. (2024). *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema* (Accessed on: 2024, 1. July)

Kovacic, I., & Honic, M. (2021). Scanning and data capturing for BIM-supported resources assessment: A case study. *Journal of Information Technology in Construction*, *26*, 624–638. https://doi.org/10.36680/j.itcon.2021.032

Moult, D. (2021, 24. July). *Introduction to OpenBIM, Native IFC, and Open Source AEC*. YouTube https://www.youtube.com/watch?v=h2Rv9iu7yDk (Accessed on: 2024, 1. July)

OpenAI. (2023). ChatGPT (version 4) [Online]. OpenAI. https://www.openai.com/

OpenCV. (2024). *Introduction.* https://docs.opencv.org/4.x/d1/dfb/intro.html (Accessed on: 2024, 1. July)

Pix4D. (2022). *Zehn Grundbegriffe der Photogrammetrie*. https://www.pix4d.com/de/blog/zehn-grundbegriffe-der-photogrammetrie/ (Accessed on: 2024, 1. July)

Ruiz Durán, C., Lemaitre, C., Braune, A., von Gemmingen, U., Schwarz, M., Jansen, F., Würfel, C., Fischer, K., & Montigel, R. (2019). *Circular economy: Closing loops means being fit for the future*. https://single-market-economy.ec.europa.eu/industry/sustainability/buildings-and-construction_en (Accessed on: 2024, 1. July)

W3Schools. (2024). *XML and Xpath.* https://www.w3schools.com/xml/xml_xpath.asp (Accessed on: 2024, 1. July)

Yang, X., Hu, M., Zhang, C., & Steubing, B. (2022). Urban mining potential to reduce primary material use and carbon emissions in the Dutch residential building sector. *Resources, Conservation and Recycling*, *180*, 106215. https://doi.org/10.1016/j.resconrec.2022.106215

Image references

Figure 5:
buildingSMART. (2024, 23. May). *6.1.3.16.2 Entity inheritance.*
https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcDoor.htm (Accessed on: 2024, July 2.)

Figure 6:
buildingSMART. (2024, 23. May). *6.1.3.16.3 Attributes*.
https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcDoor.htm (Accessed on: 2024, July 2.)

Figure 7:
buildingSMART. (2024, 23. May). *6.1.3.16.5 Property sets.*
https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcDoor.htm (Accessed on: 2024, July 2.)

Figure 8, 9, 10, 12, 13:
Concular: Bürotür, innen - Lichter Durchgang: 810 x 2100mm
https://shop.concular.de/products/23-004-eckpfeiler-immobilien-gruppe-kirschgelande-munchen-neu-23-004-ca-23-05-fovbozgfjj4fzbhhocguhb (Accessed on: 2024, July 2.))

Figure 9, 10:
Concular: Brandschutztür Wirus T30 1000x2140x170
https://shop.concular.de/products/brandschutztuer-wirus-t30-typ-1-din-r-1000x2140x170-2555 (Accessed on: 2024, July 2.)

Figure 9, 10:
Concular: Bürotür Anthrazit 1050x2150x150
https://shop.concular.de/products/buerotuer-anthrazit-din-r-1050x2150x150-6126 (Accessed on: 2024, July 2.)

List of Figures

1 Matching of data between Concular, IFC and BlenderBIM 6

2 Automating the process from Concular to BlenderBIM 6

3 Concular logo [concular.de] 8

4 BlenderBIM logo [blenderbim.org] 8

5 The class structure of IfcDoor [buildingsmart.org] 12

6 The attributes of IfcDoor [buildingsmart.org] 13

7 PropertySet templates for IfcDoor [buildingsmart.org] 14

8 Door listing in Concular [concular.de] 15

9 Descriptions of different doors from Concular (translated by the author) . . 16

10 Different descriptions of doors from Concular (translated by the author) . . 18

11 Steps of the process 19

12 Xpath of the name [concular.de] 20

13 Element of the product description [concular.de] 21

14 Steps of the image processing 23

15 BlenderBIM: Default IfcDoor in BlenderBIM 24

16 BlenderBIM: Cube on origin point 25

17 BlenderBIM: Scaled base model 26

18 BlenderBIM: UV mapping of the image onto the model 27

19 BlenderBIM: UV mapping of the other side of the model 27

20 BlenderBIM: IfcAttribute: Name; IfcAttribute: Description 28

21 BlenderBIM: IfcAttribute: Material 28

22 Function of the Add-On 29

23 BlenderBIM: Add-On and automatically created model 30

7 Appendices

```python
import bpy
import os
import re
import requests
from urllib.request import urlopen
from lxml import html
import cv2
import numpy as np
from io import BytesIO
import bmesh
from mathutils import Vector

# create Add-On
class AddProductOperator(bpy.types.Operator):
    bl_idname = "object.add_product"
    bl_label = "Add Product"

    def execute(self, context):
        entered_url = context.scene.product_url

        # main code with scraping function:
        # ensure BlenderBIM add-on is enabled
        if not bpy.context.preferences.addons.get("blenderbim"):
            bpy.ops.preferences.addon_enable(module="blenderbim")

        # create a new IFC project
        bpy.ops.bim.create_project()

        def scrape_inner_html(url, xpath):
            try:
                response = urlopen(url)
                html_content = response.read().decode('utf-8')
                tree = html.fromstring(html_content)
                elements = tree.xpath(xpath)
                data_list = []
                if elements:
                    selected_element = elements[0]
                    strong_tags = selected_element.xpath('.//strong')
                    for strong_tag in strong_tags:
                        strong_text = strong_tag.text.strip()
                        following_text = strong_tag.tail.strip() if
                            strong_tag.tail else ''
                        if following_text.startswith(':'):
                            following_text = following_text[1:].strip
                                ()
                        data_list.append((strong_text, following_text
                            ))
                return data_list
            except Exception as e:
                print(f"Error scraping {url}: {e}")
                return []

        # search for product-information
        def extract_product_name(url, name_xpath):
            try:
                response = urlopen(url)
                html_content = response.read().decode('utf-8')
                tree = html.fromstring(html_content)
                name_element = tree.xpath(name_xpath)
                if name_element:
                    product_name = name_element[0].text.strip()
                    return product_name
                else:
                    return None
            except Exception as e:
```

```
63                print(f"Error extracting product name from {url}: {e}
                      ")
64                return None
65
66            def filter_and_find_max(data_list, search_terms):
67                filtered_list = [item for item in data_list if any(term.
                      lower() in item[0].lower() for term in search_terms)]
68
69                def extract_number(value):
70                    match = re.search(r'\d+', value)
71                    return int(match.group()) if match else None
72
73                numbers = [extract_number(item[1]) for item in
                      filtered_list if extract_number(item[1]) is not None]
74                max_value = max(numbers) if numbers else None
75
76                return filtered_list, max_value
77
78            # download image and edit
79            def extract_first_image_url(url, image_xpath):
80                try:
81                    response = urlopen(url)
82                    html_content = response.read().decode('utf-8')
83                    tree = html.fromstring(html_content)
84                    image_element = tree.xpath(image_xpath)
85                    if image_element:
86                        image_url = image_element[0].get('src')
87                        if image_url.startswith('//'):
88                            image_url = 'https:' + image_url
89                        return image_url
90                    else:
91                        return None
92                except Exception as e:
93                    print(f"Error extracting image URL from {url}: {e}")
94                    return None
95
96            def download_image_to_memory(image_url):
97                try:
98                    response = requests.get(image_url.split('?')[0])
99                    image_data = BytesIO(response.content)
100                   image = cv2.imdecode(np.frombuffer(image_data.read(),
                          np.uint8), cv2.IMREAD_COLOR)
101                   print(f"Image downloaded and loaded into memory")
102                   return image
103               except Exception as e:
104                   print(f"Error downloading image from {image_url}: {e}
                          ")
105                   return None
106
107           # search for largest trapezoid to find the outline of the
                  door
108           def resize_image(image, width, height):
109               if image is None:
110                   raise ValueError("Image not found or unable to load")
111               resized_image = cv2.resize(image, (width, height),
                      interpolation=cv2.INTER_AREA)
112               return resized_image
113
114           def find_and_crop_largest_trapezoid(image,
                  output_cropped_path):
115               gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
116               blurred = cv2.GaussianBlur(gray, (5, 5), 0)
117               edged = cv2.Canny(blurred, 50, 150)
118               contours, _ = cv2.findContours(edged, cv2.RETR_LIST, cv2.
                      CHAIN_APPROX_SIMPLE)
119
120               largest_trapezoid = None
121               max_area = 0
122
123               for contour in contours:
```

```
                perimeter = cv2.arcLength(contour, True)
                approx = cv2.approxPolyDP(contour, 0.02 * perimeter,
                    True)
                if len(approx) == 4:
                    area = cv2.contourArea(approx)
                    if area > max_area:
                        max_area = area
                        largest_trapezoid = approx

            if largest_trapezoid is not None:
                x, y, w, h = cv2.boundingRect(largest_trapezoid)
                cropped = image[y:y+h, x:x+w]
                cv2.imwrite(output_cropped_path, cropped)
            else:
                cv2.imwrite(output_cropped_path, image)

        # main Execution
        url = entered_url
        xpathselector = [
            "/html/body/main/section[1]/section/div/div/div[2]/
                product-info/div[11]",
            "/html/body/main/section[1]/section/div/div/div[2]/
                product-info/div[10]",
            "/html/body/main/section[1]/section/div/div/div[2]/
                product-info/div[12]"
        ]
        name_xpath = "/html/body/main/section[1]/section/div/div/div
            [2]/product-info/div[1]/h1"
        image_xpath = "/html/body/main/section[1]/section/div/div/div
            [1]/media-gallery/slider-component[1]/ul/li[1]/div/modal-
            opener/div[2]/img"

        data_list = []
        for xpath in xpathselector:
            data_list.extend(scrape_inner_html(url, xpath))

        product_name = extract_product_name(url, name_xpath)

        terms = {'tiefe': ['tiefe'], 'l nge': ['l nge'], 'breite':
            ['breite'], 'h he': ['h he']}
        results = {}

        # check for 'tiefe' or fallback to 'l nge'
        filtered_list, max_value = filter_and_find_max(data_list,
            terms['tiefe'])
        if not filtered_list:
            filtered_list, max_value = filter_and_find_max(data_list,
                terms['l nge'])
        results['tiefe'] = (filtered_list, max_value)

        for term in ['breite', 'h he']:
            filtered_list, max_value = filter_and_find_max(data_list,
                terms[term])
            results[term] = (filtered_list, max_value)

        max_tiefe = results['tiefe'][1]
        max_breite = results['breite'][1]
        max_hoehe = results['h he'][1]

        product_description = str(data_list).strip('[]')
        values = [product_name] + [results[term][1] for term in ['
            tiefe', 'breite', 'h he']]

        # extract material information
        material_terms = [item[1] for item in data_list if 'material'
             in item[0].lower()]
        product_material = ', '.join(material_terms)
        values.append(product_material)
        values.append(product_description)

```

```
        image_save_dir = os.path.join(bpy.path.abspath("//"), "one_door")
        os.makedirs(image_save_dir, exist_ok=True)
        image_url = extract_first_image_url(url, image_xpath)
        cropped_image_path = None

        if image_url:
            image = download_image_to_memory(image_url)

            resized_image = resize_image(image, width=300, height=400)
            output_cropped_path = os.path.join(image_save_dir, "cropped_image.jpg")
            find_and_crop_largest_trapezoid(resized_image, output_cropped_path)
            cropped_image_path = output_cropped_path

        values.append(cropped_image_path)
        values.append(url)

        # Blender operations
        bpy.ops.mesh.primitive_cube_add(size=1)
        new_door = bpy.context.object

        for vert in new_door.data.vertices:
            vert.co[2] += 0.5

        # scale the object according to the extracted data
        new_door.scale = (max_tiefe/1000, max_breite/1000, max_hoehe/1000)
        new_door.name = product_name

        # add image texture
        def add_image_texture(obj, image_path):
            bpy.context.view_layer.objects.active = obj
            obj.select_set(True)

            mat = bpy.data.materials.new(name="door_image")
            mat.use_nodes = True
            nodes = mat.node_tree.nodes
            links = mat.node_tree.links

            for node in nodes:
                nodes.remove(node)

            output_node = nodes.new(type='ShaderNodeOutputMaterial')
            bsdf_node = nodes.new(type='ShaderNodeBsdfPrincipled')
            texture_node = nodes.new(type='ShaderNodeTexImage')

            texture_node.image = bpy.data.images.load(image_path)
            texture_node.extension = 'CLIP'

            links.new(bsdf_node.outputs['BSDF'], output_node.inputs['Surface'])
            links.new(texture_node.outputs['Color'], bsdf_node.inputs['Base Color'])

            if len(obj.data.materials):
                obj.data.materials[0] = mat
            else:
                obj.data.materials.append(mat)

        # set image as material
        image_path = output_cropped_path

        active_object = bpy.context.object
        add_image_texture(active_object, image_path)

        # start UV mapping
        def select_face_with_x_normal(obj, direction='positive'):
```

```
244
245             bpy.context.view_layer.objects.active = obj
246             bpy.ops.object.mode_set(mode='EDIT')
247
248             bm = bmesh.from_edit_mesh(obj.data)
249
250             for face in bm.faces:
251                 face.select = False
252
253             # UV-mapping for the largest face with the normal along
                    the x-axis
254             x_axis = Vector((1, 0, 0)) if direction == 'positive'
                    else Vector((-1, 0, 0))
255             target_face = max(bm.faces, key=lambda f: f.normal.dot(
                    x_axis))
256             target_face.select = True
257             bmesh.update_edit_mesh(obj.data)
258
259         def smart_uv_project_selected_face():
260             bpy.ops.object.mode_set(mode='EDIT')
261
262             # set UV-mapping to 'scale to bounds'
263             bpy.ops.uv.smart_project(island_margin=0.02,
                    scale_to_bounds=True)
264
265         def mirror_uv_map(constraint_axis=(True, False, False)):
266             bpy.ops.object.mode_set(mode='EDIT')
267
268             bpy.context.area.ui_type = 'UV'
269             bpy.ops.uv.select_all(action='SELECT')
270
271             bpy.ops.transform.mirror(constraint_axis=constraint_axis)
272
273         active_object = bpy.context.active_object
274         bpy.ops.object.mode_set(mode='OBJECT')
275         select_face_with_x_normal(active_object, direction='positive'
                )
276         smart_uv_project_selected_face()
277
278         # mirror the UV map along both X and Y axes
279         mirror_uv_map(constraint_axis=(True, True, False))
280         bpy.ops.object.mode_set(mode='OBJECT')
281
282         def select_face_with_x_normal(obj, direction='negative'):
283             bpy.context.view_layer.objects.active = obj
284             bpy.ops.object.mode_set(mode='EDIT')
285             bm = bmesh.from_edit_mesh(obj.data)
286             for face in bm.faces:
287                 face.select = False
288
289             x_axis = Vector((-1, 0, 0)) if direction == 'negative'
                    else Vector((1, 0, 0))
290             target_face = max(bm.faces, key=lambda f: f.normal.dot(-
                    x_axis))
291             target_face.select = True
292             bmesh.update_edit_mesh(obj.data)
293
294         def smart_uv_project_selected_face():
295             bpy.ops.object.mode_set(mode='EDIT')
296             bpy.ops.uv.smart_project(island_margin=0.02,
                    scale_to_bounds=True)
297
298         def mirror_uv_map(constraint_axis=(True, False, False)):
299             bpy.ops.object.mode_set(mode='EDIT')
300             bpy.context.area.ui_type = 'UV'
301             bpy.ops.uv.select_all(action='SELECT')
302             bpy.ops.transform.mirror(constraint_axis=constraint_axis)
303
304         active_object = bpy.context.active_object
305         bpy.ops.object.mode_set(mode='OBJECT')
```

```
306         select_face_with_x_normal(active_object, direction='positive'
                )
307         smart_uv_project_selected_face()
308         mirror_uv_map(constraint_axis=(False, True, False))
309         bpy.ops.object.mode_set(mode='OBJECT')
310 
311         # repeat UV mapping for -x-axis
312         def select_face_with_x_normal(obj, direction='negative'):
313             bpy.context.view_layer.objects.active = obj
314             bpy.ops.object.mode_set(mode='EDIT')
315             bm = bmesh.from_edit_mesh(obj.data)
316 
317             for face in bm.faces:
318                 face.select = False
319 
320             x_axis = Vector((-1, 0, 0)) if direction == 'negative'
                    else Vector((1, 0, 0))
321             target_face = max(bm.faces, key=lambda f: f.normal.dot(-
                    x_axis))
322             target_face.select = True
323             bmesh.update_edit_mesh(obj.data)
324 
325         def smart_uv_project_selected_face():
326             bpy.ops.object.mode_set(mode='EDIT')
327             bpy.ops.uv.smart_project(island_margin=0.02,
                    scale_to_bounds=True)
328 
329         # mirror only along the x-axis
330         def mirror_uv_map(constraint_axis=(True, False, False)):
331             bpy.ops.object.mode_set(mode='EDIT')
332 
333             bpy.context.area.ui_type = 'UV'
334             bpy.ops.uv.select_all(action='SELECT')
335 
336             bpy.ops.transform.mirror(constraint_axis=constraint_axis)
337 
338         active_object = bpy.context.active_object
339         bpy.ops.object.mode_set(mode='OBJECT')
340         select_face_with_x_normal(active_object, direction='positive'
                )
341         smart_uv_project_selected_face()
342         mirror_uv_map(constraint_axis=(False, True, False))
343         bpy.ops.object.mode_set(mode='OBJECT')
344 
345         bpy.context.area.ui_type = 'TEXT_EDITOR'
346 
347         # assign IFC-class
348         bpy.ops.bim.assign_class(ifc_class="IfcDoorType",
                predefined_type="DOOR", userdefined_type="")
349 
350         bpy.ops.bim.enable_editing_attributes(obj=new_door.name,
                obj_type="Object")
351         bpy.data.objects[new_door.name].BIMAttributeProperties.
                attributes[2].is_null = False
352         bpy.data.objects[new_door.name].BIMAttributeProperties.
                attributes[2].string_value = product_description
353         bpy.ops.bim.edit_attributes(obj=new_door.name, obj_type="
                Object")
354 
355         # add material
356         bpy.ops.bim.add_material(obj=new_door.name, name=
                product_material)
357 
358         bpy.context.area.ui_type = 'VIEW_3D'
359 
360         return {'FINISHED'}
361 
362 # design Add-On panel
363 class TestPanel(bpy.types.Panel):
364     bl_label = 'Add Product'
```

```
365     bl_idname = 'product_adder'
366     bl_space_type = 'VIEW_3D'
367     bl_region_type = 'UI'
368     bl_category = 'Add Product'
369 
370     def draw(self, context):
371         layout = self.layout
372         scene = context.scene
373 
374         row = layout.row()
375         row.label(text='Insert Product URL to generate')
376         row = layout.row()
377         row.prop(scene, "product_url", text="URL")
378         row = layout.row()
379         row.operator("object.add_product", text="Add")
380 
381 def register():
382     bpy.utils.register_class(AddProductOperator)
383     bpy.utils.register_class(TestPanel)
384     bpy.types.Scene.product_url = bpy.props.StringProperty(
385         name="Product URL",
386         description="Enter a product URL",
387         default=""
388     )
389 
390 def unregister():
391     bpy.utils.unregister_class(AddProductOperator)
392     bpy.utils.unregister_class(TestPanel)
393     del bpy.types.Scene.product_url
394 
395 if __name__ == '__main__':
396     register()
```